Polyhedral Compilation and Counting

Sven Verdoolaege

Polly Labs and KU Leuven
Sven.Verdoolaege@gmail.com

January 20, 2016

Outline

Outline

[3, 5, 6]

Polyhedral Compilation

Polyhedral Compilation

Analyzing and/or transforming loop programs using the *polyhedral model*

Polyhedral Model

Abstract representation of a loop program

- instance based
  - ⇒ statement *instances*
  - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
  - ⇒ integer points in unions of parametric polyhedra
  - ⇒ Presburger sets and relations
- parametric
  - ⇒ description may depend on constant symbols

[10]

Polyhedral Model

Main constituents of program representation

- Instance Set
  - $\Rightarrow$ the set of all statement instances
- Access Relations
  - $\Rightarrow$ the array elements accessed by a statement instance
- Dependences
  - $\Rightarrow$ the statement instances that depend on a statement instance
- Schedule
  - $\Rightarrow$ the relative execution order of statement instances

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Instance Set (set of statement instances)

$$\{\mathtt{S1}[i,j] : 0 \leq i < M \wedge 0 \leq j < N;$$
$$\mathtt{S2}[i,j,k] : 0 \leq i < M \wedge 0 \leq j < N \wedge 0 \leq k < K\}$$

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Instance Set (set of statement instances)

$$\{\, \mathtt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{\, \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \,\}$$
$$R = \{\, \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{A}[i,k]; \mathtt{S2}[i,j,k] \to \mathtt{B}[k,j] \,\}$$

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Instance Set (set of statement instances)

$$\{\, \mathtt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

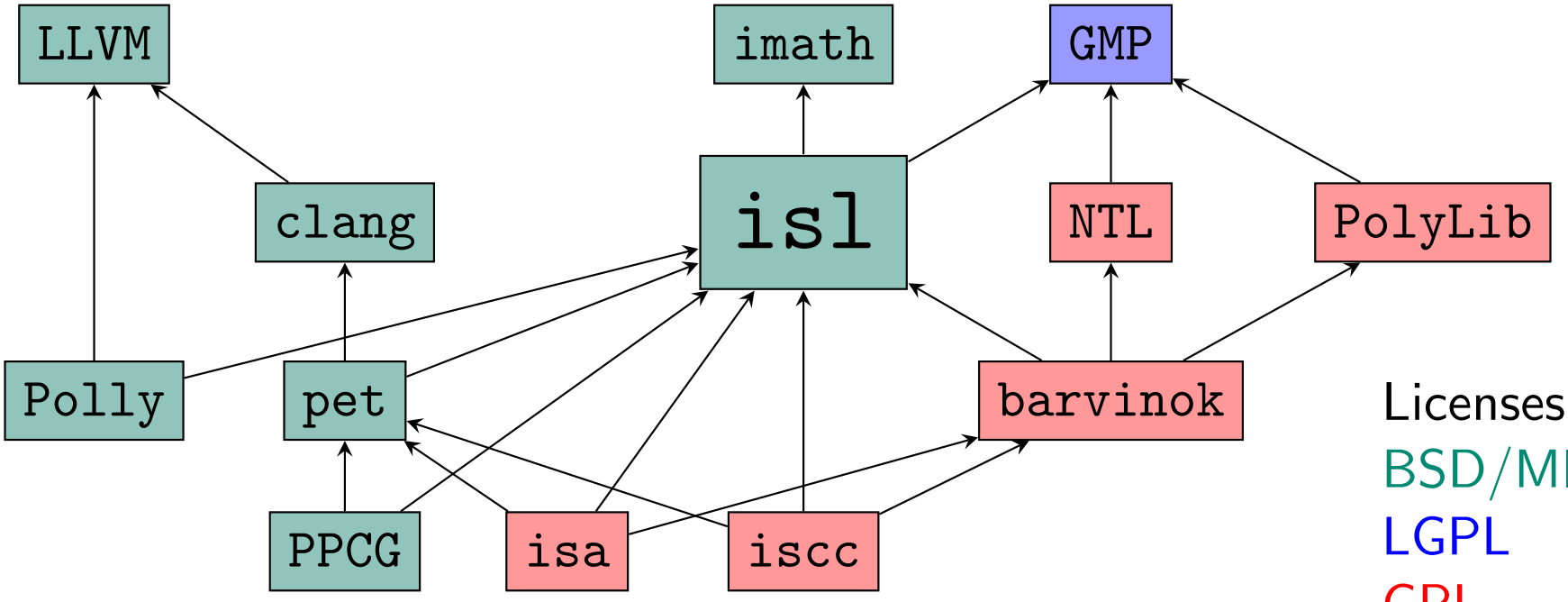
- Access Relations (accessed array elements; W: write, R: read)

$$W = \{\, \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \,\}$$
$$R = \{\, \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{A}[i,k]; \mathtt{S2}[i,j,k] \to \mathtt{B}[k,j] \,\}$$

- Schedule (relative execution order)

$$\{\, \mathtt{S1}[i,j] \to [i,j,0,0]; \mathtt{S2}[i,j,k] \to [i,j,1,k] \,\}$$

isl and Related Libraries and Tools



isl: manipulates parametric affine sets and relations

barvinok: counts elements in parametric affine sets and relations

pet: extracts polyhedral model from clang AST

PPCG: Polyhedral Parallel Code Generator

iscc: interactive calculator

isa: prototype tool set including derivation of process networks and equivalence checker

[10]

isl/iscc syntax

Relation description

[] (tuple)	`[]`
$\rightarrow$	`->`
$+, -$	`+, -`
$\cdot$	`*`
$=, \neq, \leq, <, \geq, >$	`=, !=, <=, <, >=, >`
true	`true`
false	`false`
$\wedge$	`and`
$\vee$	`or`
$\neg$	`not`
$\exists v :$	`exists v :`
$\forall v :$	`not exists v : not`
$\preccurlyeq, \prec, \succcurlyeq, \succ$	`<<=, <<, >>=, >>`

Note: constant symbols need to be explicitly declared

Outline

[10]

Basic Operations

Given two binary relations A and B

- Union (`iscc: +`)
  $A \cup B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \vee \mathbf{i} \to \mathbf{j} \in B \,\}$
- Intersection (`iscc: *`)
  $A \cap B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \wedge \mathbf{i} \to \mathbf{j} \in B \,\}$
- Difference (`iscc: -`)
  $A \setminus B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \wedge \neg(\mathbf{i} \to \mathbf{j} \in B) \,\}$

[10]

Basic Operations

Given two binary relations A and B

- Union (`iscc: +`)
  $A \cup B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \vee \mathbf{i} \to \mathbf{j} \in B \,\}$
- Intersection (`iscc: *`)
  $A \cap B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \wedge \mathbf{i} \to \mathbf{j} \in B \,\}$
- Difference (`iscc: -`)
  $A \setminus B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \wedge \neg(\mathbf{i} \to \mathbf{j} \in B) \,\}$
- Domain (`iscc: dom`)
  $\mathrm{dom}\, A = \{\, \mathbf{i} : \exists \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \,\}$

$W = \{\, \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \,\}$
$\mathrm{dom}\, W = \{\, \mathtt{S1}[i,j]; \mathtt{S2}[i,j,k] \,\}$

$\Rightarrow$ statement instances writing any array element

[10]

Basic Operations

Given two binary relations A and B

- Union (iscc: +)

  $A \cup B = \{ \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \vee \mathbf{i} \to \mathbf{j} \in B \}$
- Intersection (iscc: *)

  $A \cap B = \{ \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \wedge \mathbf{i} \to \mathbf{j} \in B \}$
- Difference (iscc: -)

  $A \setminus B = \{ \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \wedge \neg(\mathbf{i} \to \mathbf{j} \in B) \}$
- Domain (iscc: dom)

  $\mathrm{dom}\, A = \{ \mathbf{i} : \exists \mathbf{j} : \mathbf{i} \to \mathbf{j} \in A \}$

  $W = \{ \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \}$

  $\mathrm{dom}\, W = \{ \mathtt{S1}[i,j]; \mathtt{S2}[i,j,k] \}$

  $\Rightarrow$ statement instances writing any array element
- Range (iscc: ran)

  $\mathrm{ran}\, A = \{ \mathbf{j} : \exists \mathbf{i} : \mathbf{i} \to \mathbf{j} \in A \}$

  $\mathrm{ran}\, W = \{ \mathtt{C}[i,j] \}$

  $\Rightarrow$ written array elements

[10]

Domain/Range Restriction

Given two sets, A and B, and a binary relation C

- Product relation (`iscc: ->`)

  $A \to B = \{ \mathbf{i} \to \mathbf{j} : \mathbf{i} \in A \wedge \mathbf{j} \in B \}$

[10]

Domain/Range Restriction

Given two sets, A and B, and a binary relation C

- Product relation (`iscc: ->`)

  $A \to B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \in A \wedge \mathbf{j} \in B \,\}$

- Domain restriction (`iscc: *`)

  $$C \cap_{dom} A = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in C \wedge \mathbf{i} \in A \,\}$$
  $$= C \cap (A \to (\mathrm{ran}\, C))$$

[10]

Domain/Range Restriction

Given two sets, A and B, and a binary relation C

- Product relation (`iscc: ->`)

  $A \to B = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \in A \wedge \mathbf{j} \in B \,\}$

- Domain restriction (`iscc: *`)

  $C \cap_{dom} A = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in C \wedge \mathbf{i} \in A \,\}$
  $= C \cap (A \to (\mathrm{ran}\, C))$

- Range restriction (`iscc: ->*`)

  $C \cap_{ran} A = \{\, \mathbf{i} \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in C \wedge \mathbf{j} \in A \,\}$
  $= C \cap ((\mathrm{dom}\, C) \to A)$

[10]

Inverse Relation, Composition and Application

Given two binary relations A and B and set S

- Inverse (`iscc: ^-1`)

  $A^{-1} = \{ \mathbf{j} \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R \}$

  $W = \{ \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \}$
  $W^{-1} = \{ \mathtt{C}[i,j] \to \mathtt{S1}[i,j]; \mathtt{C}[i,j] \to \mathtt{S2}[i,j,k] \}$

  $\Rightarrow$ statement instances writing array element

[10]

Inverse Relation, Composition and Application

Given two binary relations A and B and set S

- Inverse (`iscc: ^-1`)

  $A^{-1} = \{ \mathbf{j} \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R \}$

  $W = \{ \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \}$
  $W^{-1} = \{ \mathtt{C}[i,j] \to \mathtt{S1}[i,j]; \mathtt{C}[i,j] \to \mathtt{S2}[i,j,k] \}$

  $\Rightarrow$ statement instances writing array element

- Composition (`iscc: after`)

  $B \circ A = \{ \mathbf{i} \to \mathbf{j} : \exists \mathbf{k} : \mathbf{i} \to \mathbf{k} \in A \wedge \mathbf{k} \to \mathbf{j} \in B \}$

  $W^{-1} \circ W = \{ \mathtt{S1}[i,j] \to \mathtt{S1}[i,j]; \mathtt{S1}[i,j] \to \mathtt{S2}[i,j,k];$
  $\mathtt{S2}[i,j,k] \to \mathtt{S1}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{S2}[i,j,k'] \}$

  $\Rightarrow$ pairs of statement instances that write same array element

[10]

Inverse Relation, Composition and Application

Given two binary relations A and B and set S

- Inverse (iscc: `^-1`)

  $A^{-1} = \{ \mathbf{j} \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R \}$

  $W = \{ \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \}$
  $W^{-1} = \{ \mathtt{C}[i,j] \to \mathtt{S1}[i,j]; \mathtt{C}[i,j] \to \mathtt{S2}[i,j,k] \}$

  $\Rightarrow$ statement instances writing array element

- Composition (iscc: `after`)

  $B \circ A = \{ \mathbf{i} \to \mathbf{j} : \exists \mathbf{k} : \mathbf{i} \to \mathbf{k} \in A \wedge \mathbf{k} \to \mathbf{j} \in B \}$

  $W^{-1} \circ W = \{ \mathtt{S1}[i,j] \to \mathtt{S1}[i,j]; \mathtt{S1}[i,j] \to \mathtt{S2}[i,j,k];$
  $\mathtt{S2}[i,j,k] \to \mathtt{S1}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{S2}[i,j,k'] \}$

  $\Rightarrow$ pairs of statement instances that write same array element

- Application (iscc: `()`)

  $A(S) = \{ \mathbf{j} : \exists \mathbf{i} \in S : \mathbf{i} \to \mathbf{j} \in A \}$

  $W(\{ \mathtt{S1}[0,j] \}) = \{ \mathtt{C}[0,j] \}$

  $\Rightarrow$ elements written by instances $\mathtt{S1}[0,j]$

Cardinality

- Cardinality of a set (`iscc: card`)
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on constant symbols

$$\mathrm{card}\, S = \{\, n : n = \#S \,\}$$

$$\mathrm{card}\, \{\, \mathtt{A}[i] : 0 \leq i \leq n; \mathtt{B}[] \,\} = n + 2$$

Cardinality

- Cardinality of a set (`iscc: card`)
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on constant symbols

$$\mathrm{card}\, S = \{\, n : n = \#S \,\}$$

$$\mathrm{card}\, \{\, \mathtt{A}[i] : 0 \leq i \leq n; \mathtt{B}[] \,\} = n + 2$$

- Cardinality of a binary relation (`iscc: card`)
  - $\Rightarrow$ for each domain element, number of corresponding images

$$\mathrm{card}\, R = \{\, \mathbf{i} \to n : n = \#\,(R(\{\, \mathbf{i} \,\})) \,\}$$

$$R = \{\, \mathtt{A}[i] \to \mathtt{C}[i] : 0 \leq i \leq n; \mathtt{B}[] \to \mathtt{C}[i] : 0 \leq i \leq n \,\}$$

$$\mathrm{card}\, R = \{\, \mathtt{A}[i] \to 1 : 0 \leq i \leq n; \mathtt{B}[] \to n + 1 \,\}$$

Cardinality

- Cardinality of a set (`iscc: card`)
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on constant symbols

$$\mathrm{card}\, S = \{\, n : n = \#S \,\}$$

$$\mathrm{card}\, \{\, \mathtt{A}[i] : 0 \leq i \leq n; \mathtt{B}[] \,\} = n + 2$$

- Cardinality of a binary relation (`iscc: card`)
  - $\Rightarrow$ for each domain element, number of corresponding images

$$\mathrm{card}\, R = \{\, \mathbf{i} \to n : n = \#\,(R(\{\, \mathbf{i} \,\})) \,\}$$

$$R = \{\, \mathtt{A}[i] \to \mathtt{C}[i] : 0 \leq i \leq n; \mathtt{B}[] \to \mathtt{C}[i] : 0 \leq i \leq n \,\}$$

$$\mathrm{card}\, R = \{\, \mathtt{A}[i] \to 1 : 0 \leq i \leq n; \mathtt{B}[] \to n + 1 \,\}$$

$\Rightarrow$ not a Presburger formula

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\mathrm{card}\{\,[i,j] : 0 \le i < N \wedge 0 \le j < N - i\,\}$

  $\Rightarrow \{\,\frac{N+N^2}{2} : N \ge 1\,\}$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\mathrm{card}\{\,[i,j] : 0 \le i < N \wedge 0 \le j < N - i\,\}$

  $\Rightarrow \{\,\frac{N+N^2}{2} : N \ge 1\,\}$
- How many times is a given array element written?

  $\mathrm{card}\left(\{\,[i,j] \to \mathrm{a}[i+j] : 0 \le i < N \wedge 0 \le j < N - i\,\}^{-1}\right)$

  $\Rightarrow \{\,\mathrm{a}[a] \to 1 + a : 0 \le a < N\,\}$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\mathrm{card}\{\,[i,j] : 0 \le i < N \wedge 0 \le j < N - i\,\}$

  $\Rightarrow \{\,\frac{N+N^2}{2} : N \ge 1\,\}$
- How many times is a given array element written?

  $\mathrm{card}\left(\{\,[i,j] \to \mathrm{a}[i+j] : 0 \le i < N \wedge 0 \le j < N - i\,\}^{-1}\right)$

  $\Rightarrow \{\,\mathrm{a}[a] \to 1 + a : 0 \le a < N\,\}$
- How many array elements are written?

  $\mathrm{card}\left(\mathrm{ran}\,\{\,[i,j] \to \mathrm{a}[i+j] : 0 \le i < N \wedge 0 \le j < N - i\,\}\right)$

  $\Rightarrow \{\,N : N \ge 1\,\}$

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Number of statement instances

$$\text{card}\{\, \mathtt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Number of statement instances

$$\text{card}\{\, \mathtt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

- Number of array elements accessed by each instance

$$\text{card}\{\, \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j];$$
$$\mathtt{S2}[i,j,k] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{A}[i,k]; \mathtt{S2}[i,j,k] \to \mathtt{B}[k,j] \,\}$$

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Number of statement instances

$$\text{card}\{\, \mathtt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

```
card [M,N,K] -> { S1[i,j] : 0 <= i < M and 0 <= j < N;
  S2[i,j,k] : 0 <= i < M and 0 <= j < N and 0 <= k < K };
```

- Number of array elements accessed by each instance

$$\text{card}\{\, \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j];$$
$$\mathtt{S2}[i,j,k] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{A}[i,k]; \mathtt{S2}[i,j,k] \to \mathtt{B}[k,j] \,\}$$

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

How many instances of S and T are executed by f2?

How many array elements accessed by each instance?

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

How many instances of S and T are executed by f2? `m>n ? m*n-n+m : m*n`

How many array elements accessed by each instance?

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

How many instances of S and T are executed by f2? `m>n ? m*n-n+m : m*n`

How many array elements accessed by each instance? `S[i]->1;T[i,j]->1+j`

Cardinality Examples (2)

How many times is S1 executed ?

```
for (i = max(0,N-M); i <= N-M+3; i++)
    for (j = 0; j <= N-2*i; j++)
        S1;
```

$$\mathrm{card}\{\,[i,j] : 0, N-M \le i \le N-M+3 \wedge 0 \le j \le N-2i\,\}$$

$$\begin{cases} -4N+8M-8 & \text{if } M \le N \le 2M-6 \\ MN-2N-M^2+6M-8 & \text{if } N \le M \le N+3 \wedge N \le 2M \\ \frac{N^2}{4}+\frac{3}{4}N+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } 0 \le N \le M \wedge 2M \le N+6 \\ \frac{N^2}{4}-MN-\frac{5}{4}N+M^2+2M+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } M \le N \le 2M \le N+6 \end{cases}$$

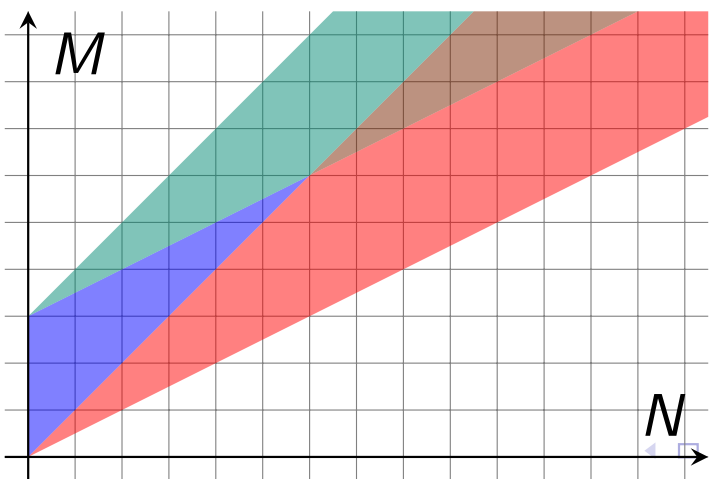
Cardinality Examples (2)

How many times is S1 executed ?

```
for (i = max(0,N-M); i <= N-M+3; i++)
    for (j = 0; j <= N-2*i; j++)
        S1;
```

$$\mathrm{card}\{\,[i,j] : 0, N-M \le i \le N-M+3 \wedge 0 \le j \le N-2i\,\}$$

$$\begin{cases} -4N+8M-8 & \text{if } M \le N \le 2M-6 \\ MN-2N-M^2+6M-8 & \text{if } N \le M \le N+3 \wedge N \le 2M \\ \frac{N^2}{4}+\frac{3}{4}N+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } 0 \le N \le M \wedge 2M \le N+6 \\ \frac{N^2}{4}-MN-\frac{5}{4}N+M^2+2M+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } M \le N \le 2M \le N+6 \end{cases}$$



Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$

  $\Rightarrow$ Presburger term

  That is, a term constructed from variables, constant symbols, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, constant symbols, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, constant symbols, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, constant symbols, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, constant symbols, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, constant symbols, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, constant symbols, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
  - $\Rightarrow$ a list of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, constant symbols, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, constant symbols, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
  - $\Rightarrow$ a list of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

Note: in practice, cardinality result has no nested integer divisions

Basic Operations on Piecewise Expressions

- Piecewise (rational) quasi affine expressions
  - addition $(+)$
  - subtraction $(-)$
  - negation $(-)$
  - minimum (min), maximum (max)
  - multiplication by constant $(\cdot d)$
  - division by constant $(/d)$
  - remainder on integer division by constant $(\bmod d)$
  - floor $(\lfloor \cdot \rfloor)$
  - ceiling $(\lceil \cdot \rceil)$

Basic Operations on Piecewise Expressions

- Piecewise (rational) quasi affine expressions
  - addition $(+)$
  - subtraction $(-)$
  - negation $(-)$
  - minimum (min), maximum (max)
  - multiplication by constant $(\cdot d)$
  - division by constant $(/d)$
  - remainder on integer division by constant $(\bmod d)$
  - floor $(\lfloor \cdot \rfloor)$
  - ceiling $(\lceil \cdot \rceil)$
- Piecewise quasi polynomial expressions
  - addition $(+)$
  - subtraction $(-)$
  - negation $(-)$
  - multiplication $(\cdot)$
  - exponentiation by positive integer constant $(\cdot^d)$

[10]

Spaces

The elements of a set are of the form

- $n[i_0, i_1, \ldots, i_{d-1}]$, with S1[0, 1]
  - n an identifier
  - $d \geq 0$
  - i_j integers

Define

Space The space $\mathcal{S}\mathbf{i}$ of an element $\mathbf{i}$ is

- n/d S1/2

Values The value vector $\mathcal{V}\mathbf{i}$ of an element $\mathbf{i}$ is

- $(i_0, i_1, \ldots, i_{d-1})$ (0, 1)

Lexicographic Order

[10]

Given two integer vectors, $\mathbf{a}$ and $\mathbf{b}$, two sets, S and T, and two binary relations, A and B

- Integer vectors

$$\mathbf{a} \prec \mathbf{b} \coloneqq \bigvee_{i:1 \leq i \leq n} \left(\left(\bigwedge_{j:1 \leq j < i} a_j = b_j \right) \wedge a_i < b_i \right)$$

Lexicographic Order

[10]

Given two integer vectors, **a** and **b**, two sets, S and T, and two binary relations, A and B

- Integer vectors

$$\mathbf{a} \prec \mathbf{b} \coloneqq \bigvee_{i:1 \leq i \leq n} \left(\left(\bigwedge_{j:1 \leq j < i} a_j = b_j \right) \wedge a_i < b_i \right)$$

- Sets (`iscc: <<`)

$$S \prec T = \{ \mathbf{i} \to \mathbf{j} : \mathbf{i} \in S \wedge \mathbf{j} \in T \wedge \mathcal{S}\mathbf{i} = \mathcal{S}\mathbf{j} \wedge \mathcal{V}\mathbf{i} \prec \mathcal{V}\mathbf{j} \}$$

Lexicographic Order

[10]

Given two integer vectors, $\mathbf{a}$ and $\mathbf{b}$, two sets, S and T, and two binary relations, A and B

- Integer vectors

$$\mathbf{a} \prec \mathbf{b} \coloneqq \bigvee_{i:1 \leq i \leq n} \left(\left(\bigwedge_{j:1 \leq j < i} a_j = b_j \right) \wedge a_i < b_i \right)$$

- Sets (`iscc: <<`)

$$S \prec T = \{ \mathbf{i} \to \mathbf{j} : \mathbf{i} \in S \wedge \mathbf{j} \in T \wedge \mathcal{S}\mathbf{i} = \mathcal{S}\mathbf{j} \wedge \mathcal{V}\mathbf{i} \prec \mathcal{V}\mathbf{j} \}$$

- Binary relations (`iscc: <<`)

  $\Rightarrow$ binary relation on domains reflecting lexicographic order of images

$$A \prec B = \{ \mathbf{a} \to \mathbf{b} : \exists \mathbf{c}, \mathbf{d} : \mathbf{a} \to \mathbf{c} \in A \wedge \mathbf{b} \to \mathbf{d} \in B \wedge \mathcal{S}\mathbf{c} = \mathcal{S}\mathbf{d} \wedge \mathcal{V}\mathbf{c} \prec \mathcal{V}\mathbf{d} \}$$

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
S := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
S << S;
```

Illustrative Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
S := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
S << S;
```

```
{ S2[i, j, k] -> S2[i', j', k'] : i' >= 1 + i;
  S2[i, j, k] -> S2[i, j', k'] : j' >= 1 + j;
  S2[i, j, k] -> S2[i, j, k'] : k' >= 1 + k;
  S1[i, j] -> S2[i', j', k] : i' >= 1 + i;
  S1[i, j] -> S2[i, j', k] : j' >= 1 + j;
  S1[i, j] -> S2[i, j, k]; S2[i, j, k] -> S1[i', j'] : i' >= 1 + i;
  S2[i, j, k] -> S1[i, j'] : j' >= 1 + j;
  S1[i, j] -> S1[i', j'] : i' >= 1 + i;
  S1[i, j] -> S1[i, j'] : j' >= 1 + j }
```

[10]

Lexicographic Optimization

- Space-local operation
  - Decompose set/relation along spaces (S_i/R_i)
  - Apply operation on each subset
  - Take union of results

[10]

Lexicographic Optimization

- Space-local operation
  - Decompose set/relation along spaces (S_i/R_i)
  - Apply operation on each subset
  - Take union of results
- Lexicographic Minimum of Sets (`iscc: lexmin`)

$$\text{lexmin}\, S_i = \{\, \mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : \mathcal{V}\mathbf{x} \preccurlyeq \mathcal{V}\mathbf{y} \,\}$$

$$I = \{\, \mathtt{R}[]; \mathtt{S}[i,j] : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}[k] : 0 \le k < 2 \,\}$$

$$\text{lexmin}\, I = \{\, \mathtt{R}[]; \mathtt{S}[0,0]; \mathtt{T}[0] \,\}$$

[10]

Lexicographic Optimization

- Space-local operation
  - Decompose set/relation along spaces (S_i/R_i)
  - Apply operation on each subset
  - Take union of results
- Lexicographic Minimum of Sets (`iscc: lexmin`)

  $\text{lexmin}\, S_i = \{\, \mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : \mathcal{V}\mathbf{x} \preccurlyeq \mathcal{V}\mathbf{y} \,\}$

  $I = \{\, \mathtt{R}[]; \mathtt{S}[i,j] : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}[k] : 0 \le k < 2 \,\}$

  $\text{lexmin}\, I = \{\, \mathtt{R}[]; \mathtt{S}[0,0]; \mathtt{T}[0] \,\}$
- Lexicographic Maximum of Sets (`iscc: lexmax`)

  $\text{lexmax}\, S_i = \{\, \mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : \mathcal{V}\mathbf{x} \succcurlyeq \mathcal{V}\mathbf{y} \,\}$

  $\text{lexmax}\, I = \{\, \mathtt{R}[]; \mathtt{S}[1,1]; \mathtt{T}[1] \,\}$

[10]

Lexicographic Optimization

- Space-local operation
  - Decompose set/relation along spaces (S_i/R_i)
  - Apply operation on each subset
  - Take union of results
- Lexicographic Minimum of Sets (iscc: `lexmin`)

  $\text{lexmin}\, S_i = \{\, \mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : \mathcal{V}\mathbf{x} \preccurlyeq \mathcal{V}\mathbf{y} \,\}$

  $I = \{\, \mathtt{R}[]; \mathtt{S}[i,j] : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}[k] : 0 \le k < 2 \,\}$

  $\text{lexmin}\, I = \{\, \mathtt{R}[]; \mathtt{S}[0,0]; \mathtt{T}[0] \,\}$
- Lexicographic Maximum of Sets (iscc: `lexmax`)

  $\text{lexmax}\, S_i = \{\, \mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : \mathcal{V}\mathbf{x} \succcurlyeq \mathcal{V}\mathbf{y} \,\}$

  $\text{lexmax}\, I = \{\, \mathtt{R}[]; \mathtt{S}[1,1]; \mathtt{T}[1] \,\}$
- Lexicographic Maximum of Relations (iscc: `lexmax`)

  $\text{lexmax}\, R_i = \{\, \mathbf{x} \to \mathbf{y} : \mathbf{x} \to \mathbf{y} \in R_i \wedge \forall \mathbf{x}' \to \mathbf{z} \in R_i : \mathbf{x} = \mathbf{x}' \Rightarrow \mathcal{V}\mathbf{y} \succcurlyeq \mathcal{V}\mathbf{z} \,\}$

  $W = \{\, [i,j] \to \mathrm{a}[i+j] : 0 \le i < N \wedge 0 \le j < N - i \,\}$

  $\text{lexmax}(W^{-1}) = \{\, \mathrm{a}[a] \to [a,0] : 0 \le a < N \,\}$

  $\Rightarrow$ last statement instance writing array element

[1]

Reuse Distance Computation

Given an access to a cache line ℓ, how many distinct cache lines have been accessed since the previous access to ℓ?

$\Rightarrow$ Is the cache line still in the cache?

[1]

Reuse Distance Computation

Given an access to a cache line ℓ, how many distinct cache lines have been accessed since the previous access to ℓ?

$\Rightarrow$ Is the cache line still in the cache?

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```

Assume `A[i]` in cache line $\lfloor i/3 \rfloor$

[1]

Reuse Distance Computation

Given an access to a cache line ℓ, how many distinct cache lines have been accessed since the previous access to ℓ?

$\Rightarrow$ Is the cache line still in the cache?

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```

Assume `A[i]` in cache line $\lfloor i/3 \rfloor$

i	0			1			2			3			4		5		6		7	
r	a	b	c	a	b	c	a	b	c	a	b	c	a	b	a	b	a	b	a	b
$r@i$	0	7	0	1	6	2	2	5	4	3	4	6	4	3	5	2	6	1	7	0
$\lfloor (r@i)/3 \rfloor$	0	2	0	0	2	0	0	1	1	1	1	2	1	1	1	0	2	0	2	0
distance	0	0	2	1	2	2	1	0	1	1	1	3	2	1	1	3	3	2	2	2

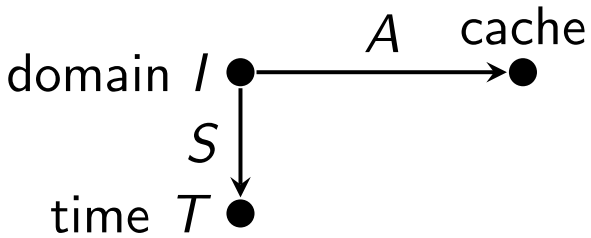
Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```

Assume `A[i]` in cache line $\lfloor i/3 \rfloor$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i, 0]; \mathtt{b}[i] \to [i, 1]; \mathtt{c}[i] \to [i, 2] \}$$

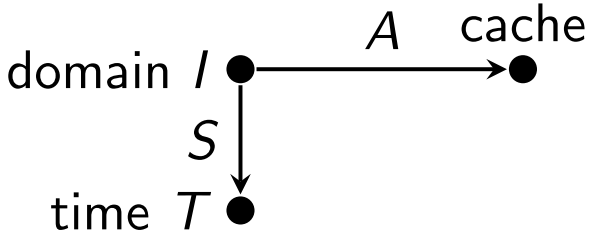
$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i, 0]; \mathtt{b}[i] \to [i, 1]; \mathtt{c}[i] \to [i, 2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap (T \prec T) \right) \right) \circ S$$

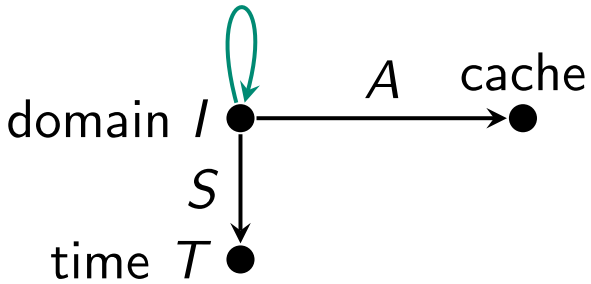
$$before = S^{-1} \circ (T \succcurlyeq T) \circ S$$

$$after\_prev = before^{-1} \circ next^{-1}$$

$$distance = \operatorname{card} (A \circ (after\_prev \cap before))$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i, 0]; \mathtt{b}[i] \to [i, 1]; \mathtt{c}[i] \to [i, 2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap (T \prec T) \right) \right) \circ S$$

$$before = S^{-1} \circ (T \prec T) \circ S$$

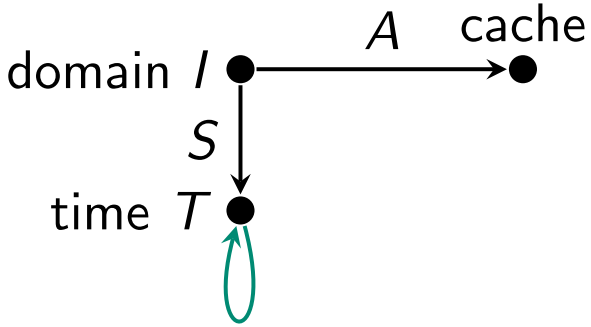
pair of statement instances that access same cache line

$$after\_prev = before^{-1} \circ next^{-1}$$

$$distance = \operatorname{card} (A \circ (after\_prev \cap before))$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i,0]; \mathtt{b}[i] \to [i,1]; \mathtt{c}[i] \to [i,2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap (T \prec T) \right) \right) \circ S$$

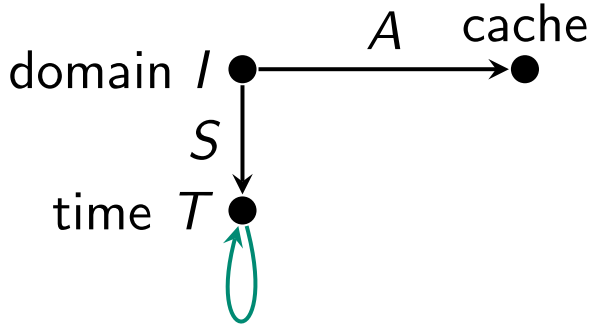
$before = S^{-1} \circ (T \prec T) \circ S$ schedule times of pair of statement instances that access same cache line

$after\_prev = before^{-1} \circ next^{-1}$

$$distance = \operatorname{card} (A \circ (after\_prev \cap before))$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i,0]; \mathtt{b}[i] \to [i,1]; \mathtt{c}[i] \to [i,2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap \left(T \prec T \right) \right) \right) \circ S$$

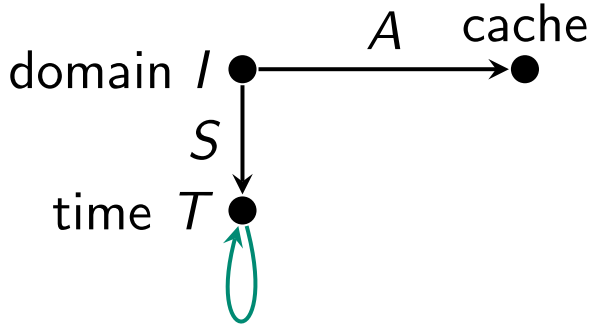
before schedule times of pair of statement instances that access same cache line such that first is executed before second

after_prev $= before^{-1} \circ next$

$$distance = \operatorname{card} \left(A \circ \left(after\_prev \cap before \right) \right)$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i,0]; \mathtt{b}[i] \to [i,1]; \mathtt{c}[i] \to [i,2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap \left(T \prec T \right) \right) \right) \circ S$$

schedule times of a statement instance and the next statement instance that accesses same cache line

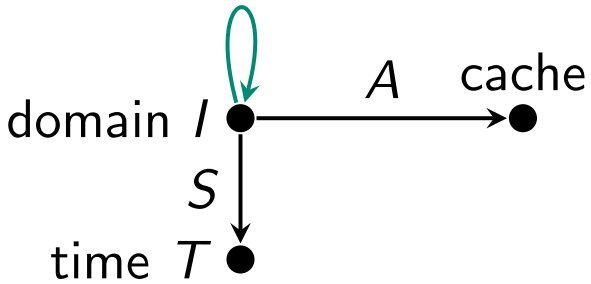
$$before = S^{-1} \circ (T \prec T) \circ S$$

$$after\_prev = before^{-1} \circ next$$

$$distance = \operatorname{card} \left(A \circ \left(after\_prev \cap before \right) \right)$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i, 0]; \mathtt{b}[i] \to [i, 1]; \mathtt{c}[i] \to [i, 2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap (T \prec T) \right) \right) \circ S$$

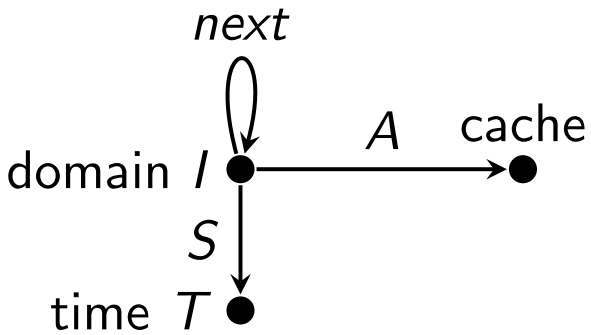
$before$ a statement instance and the next statement instance that accesses same cache line

$$after\_prev = before^{-1} \circ next^{-1}$$

$$distance = \operatorname{card} (A \circ (after\_prev \cap before))$$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {
a:   A[i];
b:   A[7-i];
     if (i <= 3)
c:       A[2*i];
}
```



Assume A[i] in cache line $\lfloor i/3 \rfloor$

$$I = \{ \mathtt{a}[i] : 0 \le i \le 7; \mathtt{b}[i] : 0 \le i \le 7; \mathtt{c}[i] : 0 \le i \le 3; \}$$

$$R = \{ \mathtt{a}[i] \to \mathtt{A}[i]; \mathtt{b}[i] \to \mathtt{A}[7-i]; \mathtt{c}[i] \to \mathtt{A}[2i] \} \cap_{dom} I$$

$$S = \{ \mathtt{a}[i] \to [i,0]; \mathtt{b}[i] \to [i,1]; \mathtt{c}[i] \to [i,2] \}$$

$$C = \{ \mathtt{A}[a] \to \mathtt{L}[\lfloor a/3 \rfloor] \}$$

$$A = C \circ R$$

$$T = \operatorname{ran} S$$

$$next = S^{-1} \circ \left(\operatorname{lexmin} \left(\left(S \circ \left(A^{-1} \circ A \right) \circ S^{-1} \right) \cap (T \prec T) \right) \right) \circ S$$

$$before = S^{-1} \circ (T \succcurlyeq T) \circ S$$

$$after\_prev = before^{-1} \circ next^{-1}$$

$$distance = \operatorname{card} (A \circ (after\_prev \cap before))$$

Outline

Bounds on Piecewise Quasi Polynomials

$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Bounds on Piecewise Quasi Polynomials

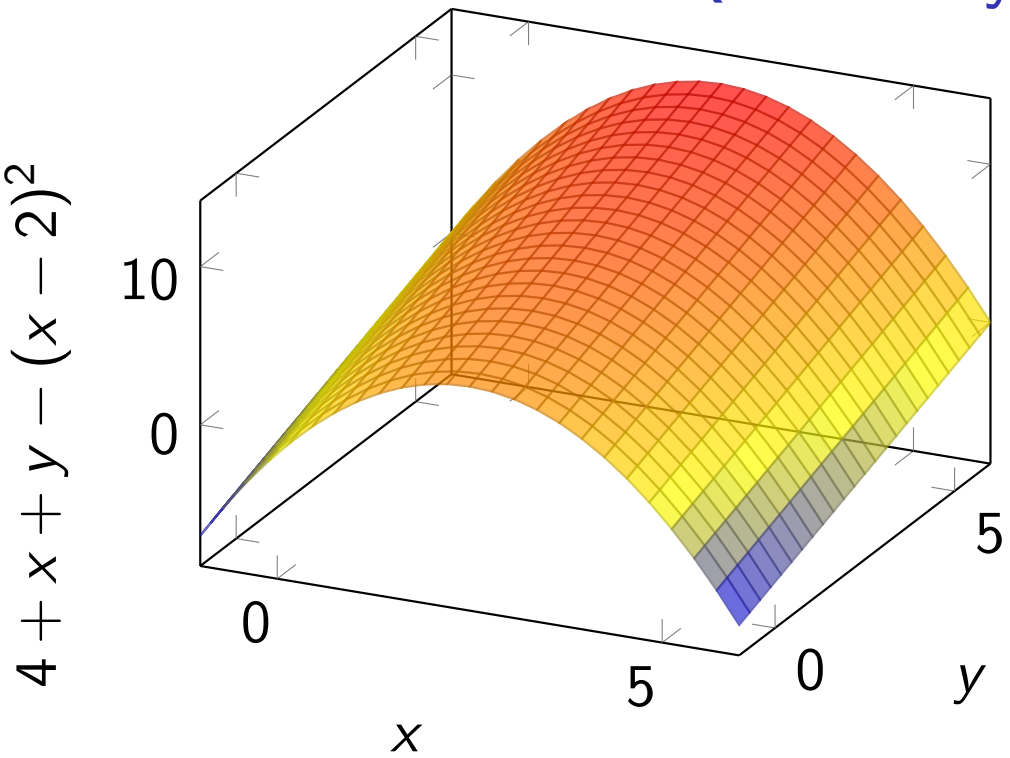
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



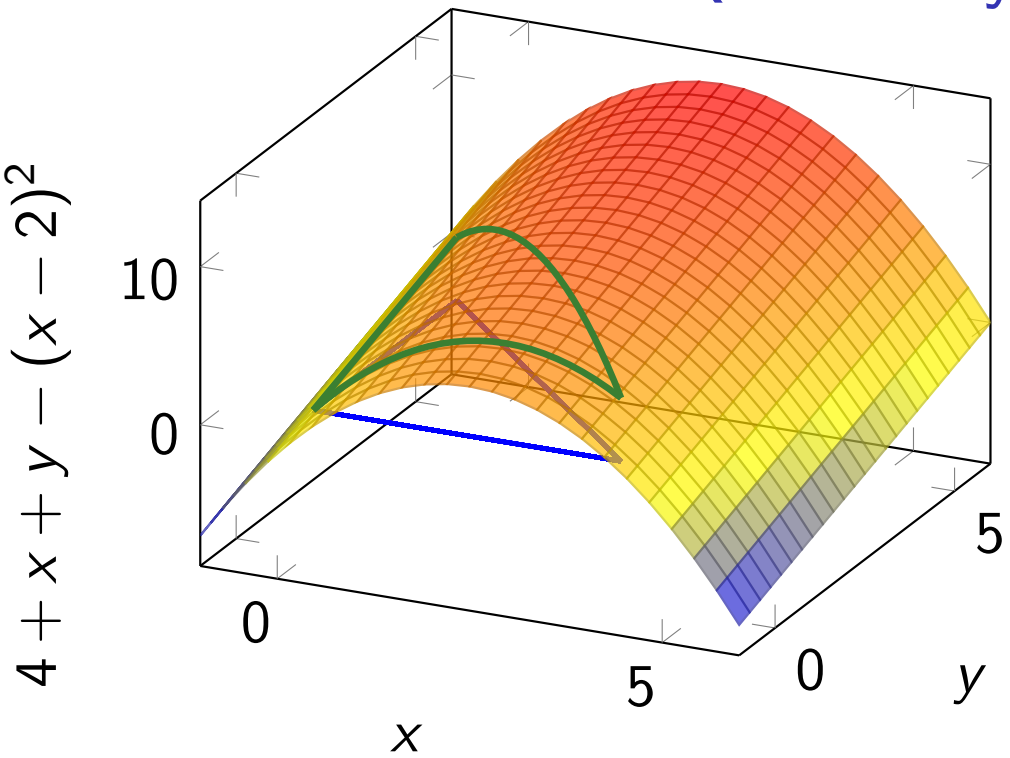
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

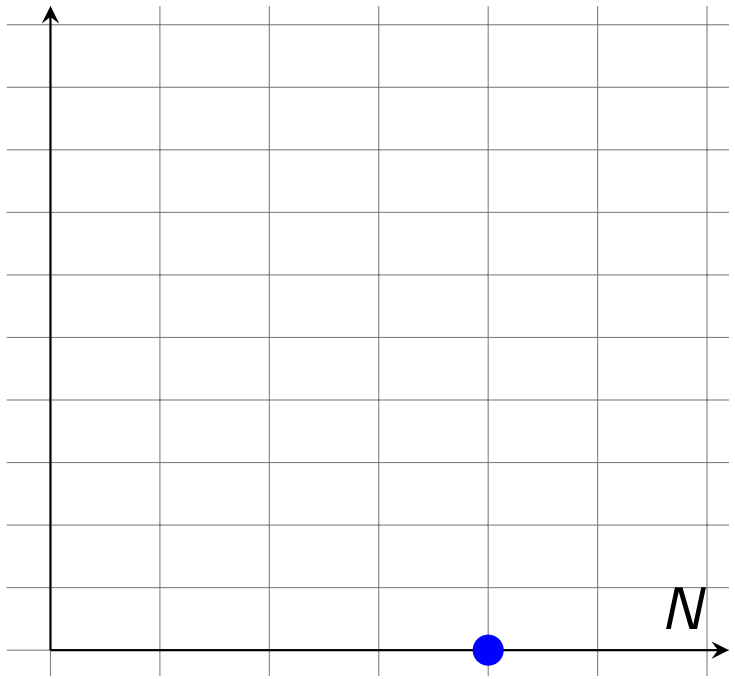
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





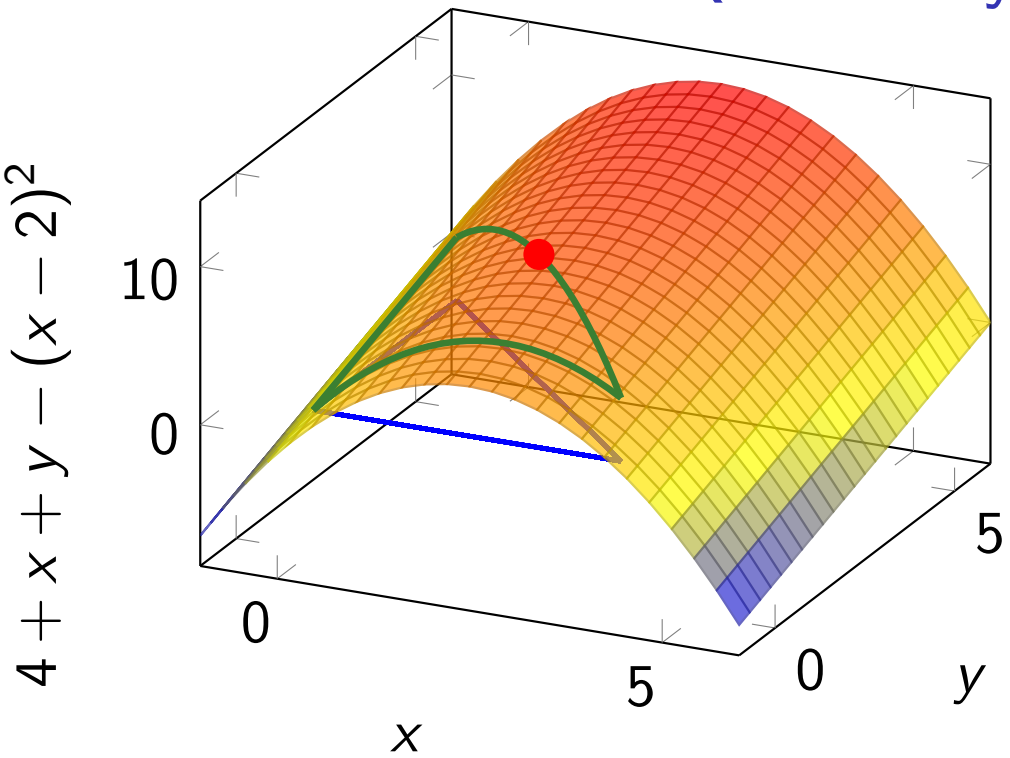
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

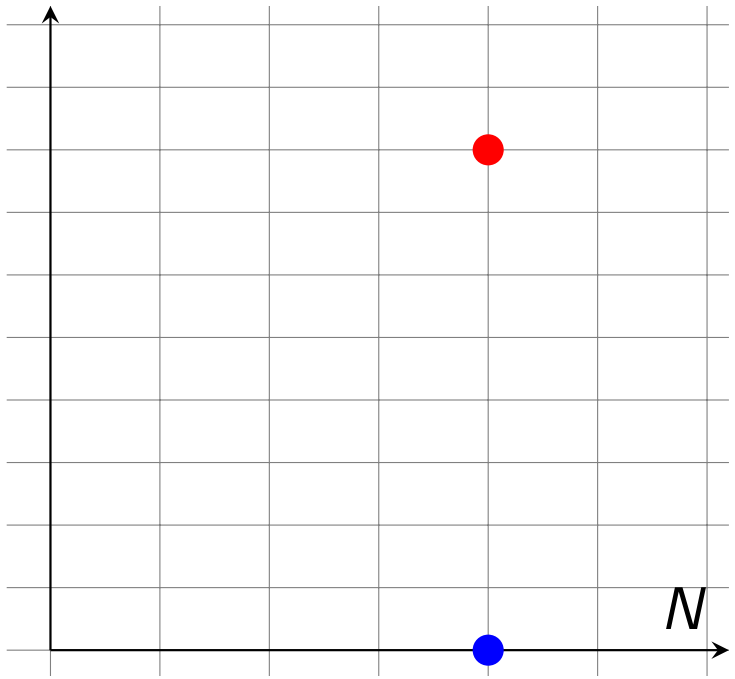
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





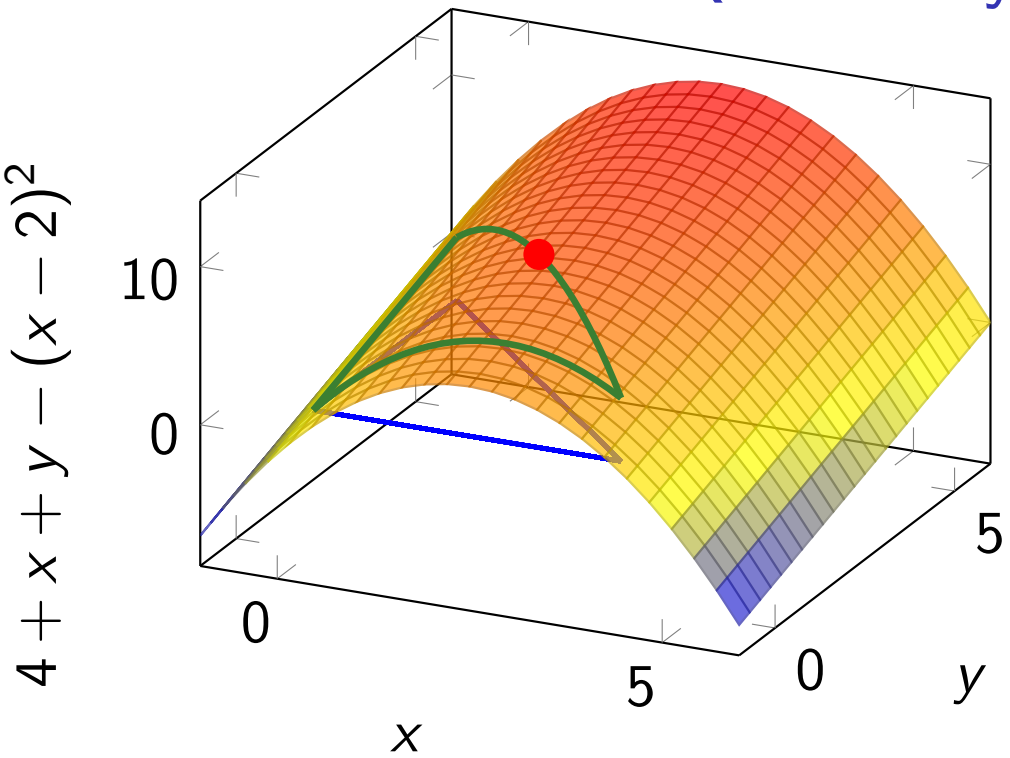
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

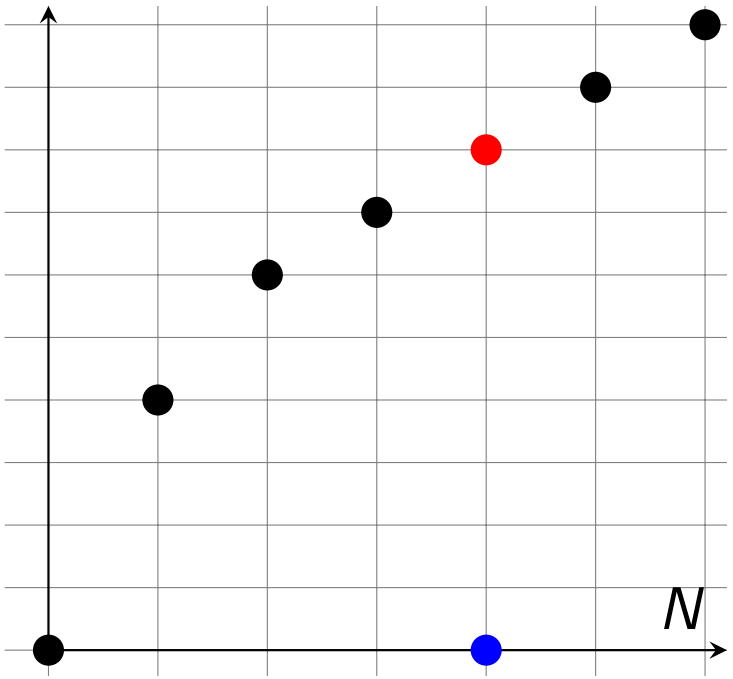
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





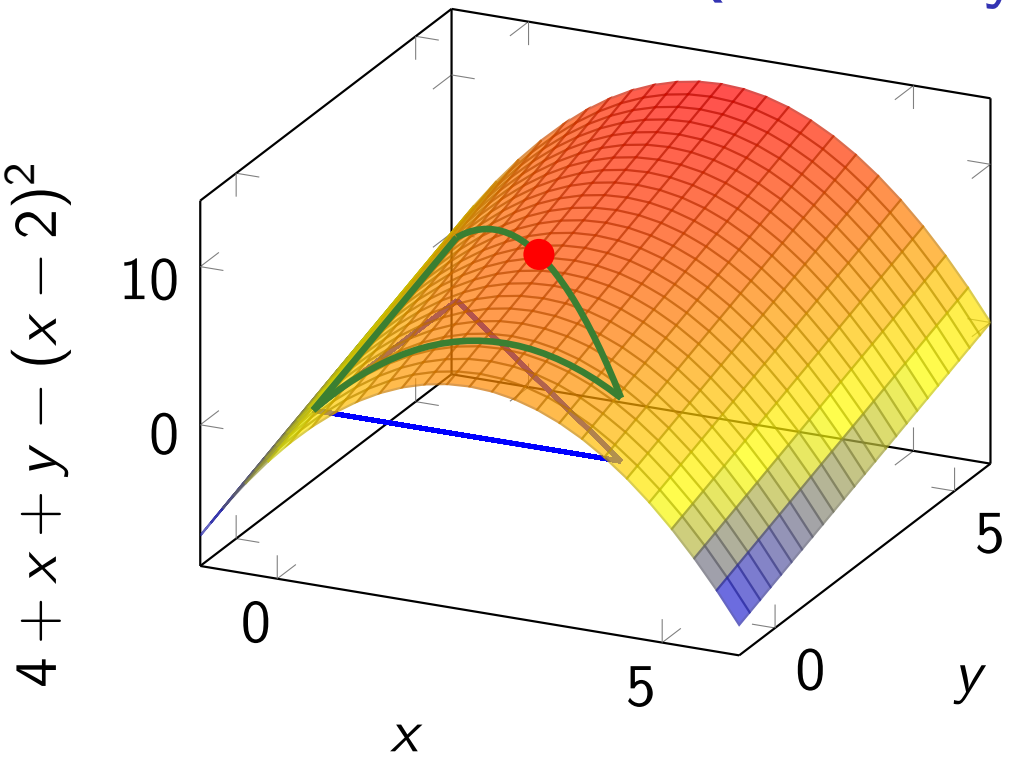
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

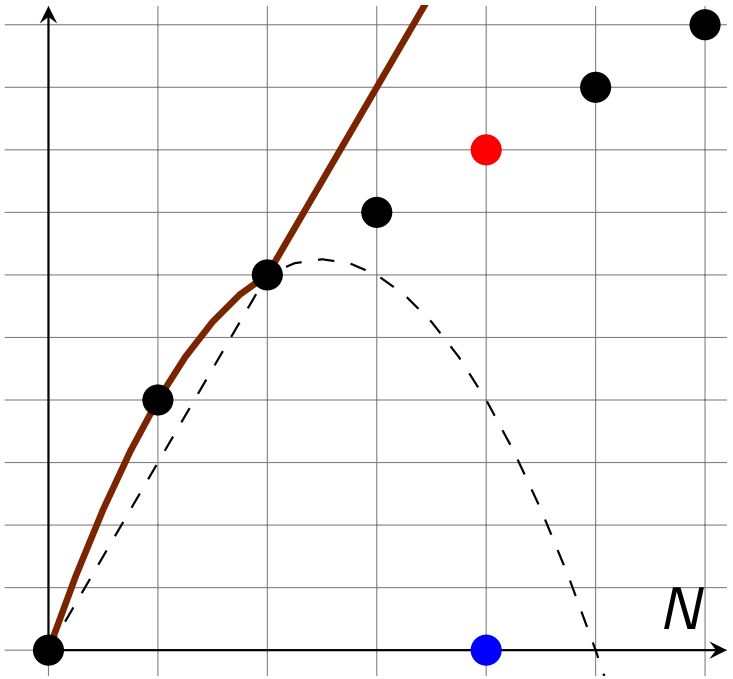
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2 \leq u(N) = \max(3N, 5N - N^2)$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\mathrm{ub}\begin{cases} ij + i - N + 1 & \text{if } 0 \leq i < N \wedge i \leq j < N \end{cases}$$

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\mathrm{ub}\begin{cases} ij + i - N + 1 & \text{if } 0 \le i < N \wedge i \le j < N \end{cases}$$

Result:

$$\begin{cases} \max(1 - 2N + N^2) & \text{if } N \ge 1 \end{cases}$$

(exact maximum)

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\mathrm{ub}\begin{cases} ij + i - N + 1 & \text{if } 0 \le i < N \wedge i \le j < N \end{cases}$$

```
ub [N] -> {[i,j] -> i*j+i-N+1: 0 <= i < N and i <= j < N};
```

Result:

$$\begin{cases} \max(1 - 2N + N^2) & \text{if } N \ge 1 \end{cases}$$

(exact maximum)

Maximal Number of Live Memory elements

- Assume each statement instance writes to at most one array element
  $\Rightarrow$ Each live element can be identified by write instance
- Compute dataflow relation D
  $\Rightarrow$ Pairs of write instances and corresponding read instances
- For each write instance compute last read
  $L = S^{-1} \circ \mathrm{lexmax}(S \circ D)$
- For each statement instance **i**, count write instances that precede **i** such that corresponding last read follows **i**
  $\Rightarrow$ Number of live elements at **i**
  $N = \mathrm{card}\left(((S \succ S) \cap_{ran} (\mathrm{dom}\, L)) \cap \left(L^{-1} \circ (S \preccurlyeq S)\right)\right)$
- Compute upper bound
  $U = \mathrm{ub}\, N$

Maximal Number of Live Memory elements — Example

```
for (i = 0; i < N; ++i)
S1:     t[i] = f(a[i]);
for (i = 0; i < N; ++i)
S2:     b[i] = g(t[N-i-1]);
```

$$I = \{ \mathrm{S1}[i] : 0 \leq i < N; \mathrm{S2}[i] : 0 \leq i < N \}$$

$$S = \{ \mathrm{S1}[i] \rightarrow [0, i]; \mathrm{S2}[i] \rightarrow [1, i] \}$$

$$D = \{ \mathrm{S1}[i] \rightarrow \mathrm{S2}[N - 1 - i] : 0 \leq i < N \}$$

Maximal Number of Live Memory elements — Example

```
for (i = 0; i < N; ++i)
S1:     t[i] = f(a[i]);
for (i = 0; i < N; ++i)
S2:     b[i] = g(t[N-i-1]);
```

$$I = \{\, \mathrm{S1}[i] : 0 \le i < N; \mathrm{S2}[i] : 0 \le i < N \,\}$$

$$S = \{\, \mathrm{S1}[i] \to [0, i]; \mathrm{S2}[i] \to [1, i] \,\}$$

$$D = \{\, \mathrm{S1}[i] \to \mathrm{S2}[N - 1 - i] : 0 \le i < N \,\}$$

$$L = S^{-1} \circ \mathrm{lexmax}(S \circ D)$$

$$= \{\, \mathrm{S1}[i] \to \mathrm{S2}[N - 1 - i] : 0 \le i < N \,\}$$

$$N = \mathrm{card}\left(((S \succ S) \cap_{ran} (\mathrm{dom}\, L)) \cap \left(L^{-1} \circ (S \preccurlyeq S)\right)\right)$$

$$= \{\, \mathrm{S1}[i] \to i : 1 \le i < N; \mathrm{S2}[i] \to N - i : 0 \le i < N \,\}$$

$$U = \mathrm{ub}\, N$$

$$= \{\, \mathrm{max}(N) : N \ge 1 \,\}$$

Outline

Incremental Counting

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- direct computation

```
card [N] -> { [i,j] : 0<=i<N and 0<=j<N-i };
```

Incremental Counting

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- direct computation

  ```
  card [N] -> { [i,j] : 0<=i<N and 0<=j<N-i };
  ```

- incremental computation

  ```
  card [N] -> { [i] -> [j] : 0<=i<N and 0<=j<N-i };
  ```

Incremental Counting

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- direct computation

  ```
  card [N] -> { [i,j] : 0<=i<N and 0<=j<N-i };
  ```

- incremental computation

  ```
  card [N] -> { [i] -> [j] : 0<=i<N and 0<=j<N-i };
  ```

  Result:

  ```
  [N] -> { [i] -> (N - i) : i <= -1 + N and i >= 0 }
  ```

  ```
  sum [N] -> { [i] -> (N - i) : i <= -1 + N and i >= 0 };
  ```

  $\Rightarrow$ sum over all elements in domain

Weighted Counting

$$G = F \circ R$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$

Weighted Counting

$$G = F \circ R = \left\{ [x, y] \to \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ [x] \to [x, y] \}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

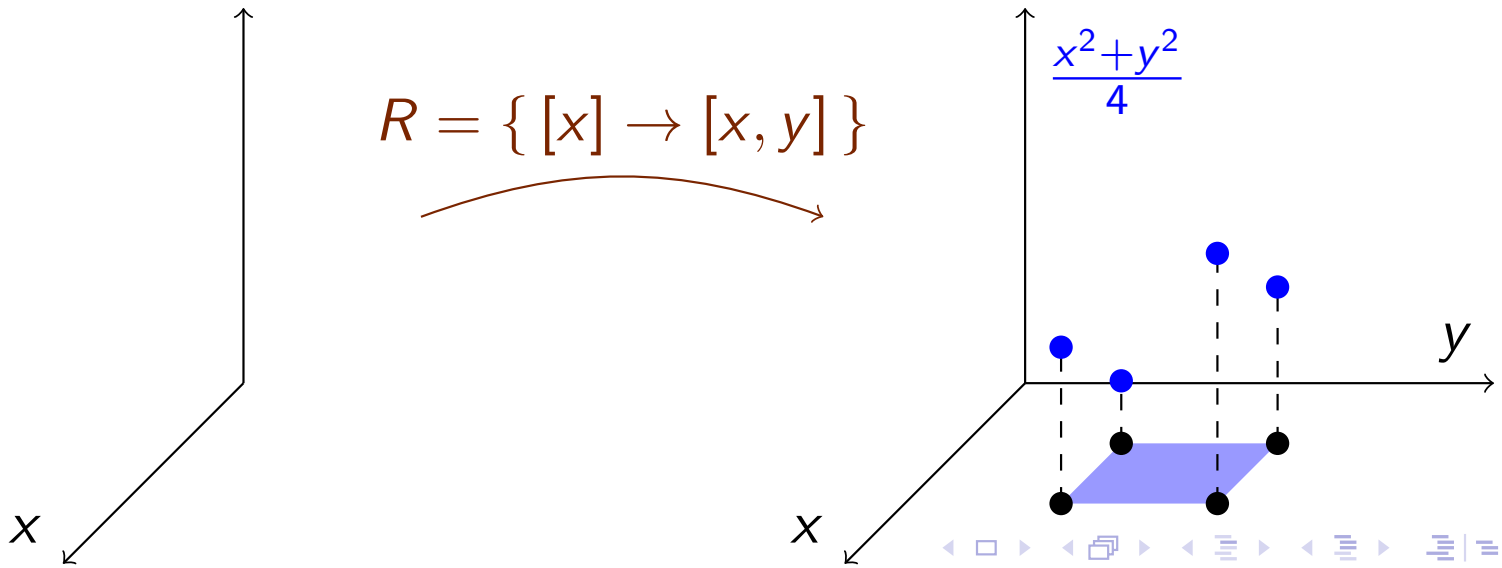
$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$

Weighted Counting

$$G = F \circ R = \left\{ [x, y] \to \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ [x] \to [x, y] \}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$



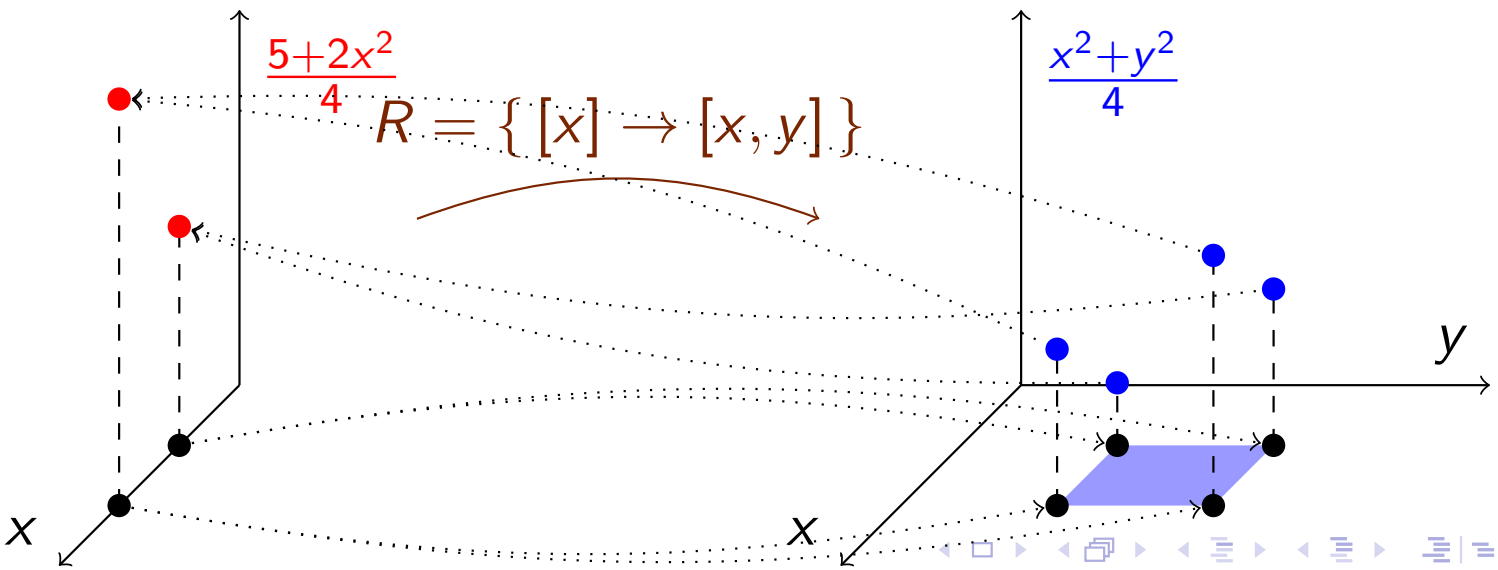
Weighted Counting

$$G = F \circ R = \left\{ [x,y] \to \frac{x^2+y^2}{4} : 1 \le x,y \le 2 \right\} \circ \{ [x] \to [x,y] \}$$

$$= \left\{ [x] \to \frac{5+2x^2}{2} : 1 \le x \le 2 \right\}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$



Compositions with Piecewise (Folds of) Quasi polynomials

$$F \circ R$$

- R: $D_1 \to D_2$ is a Presburger relation
- F: $D_2 \to \mathbb{Q}$ may be
  - piecewise quasi polynomial
    (result of counting problem)

    $\Rightarrow$ take sum over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$
  - piecewise fold of quasi polynomials
    (result of upper bound computation)

    $\Rightarrow$ compute bound over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$
- $(F \circ R)$: $D_1 \to \mathbb{Q}$ of same type as F

if R is single-valued, then sum/bound is computed over a single point

Compositions with Piecewise (Folds of) Quasi polynomials

$$F \circ R \qquad \text{or} \qquad F(S)$$

- R: $D_1 \to D_2$ is a Presburger relation
- $S \subseteq D_2$ is a Presburger set
- F: $D_2 \to \mathbb{Q}$ may be
  - piecewise quasi polynomial
    (result of counting problem)

    $\Rightarrow$ take sum over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$ or $S \cap (\operatorname{dom} F)$
  - piecewise fold of quasi polynomials
    (result of upper bound computation)

    $\Rightarrow$ compute bound over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$ or $S \cap (\operatorname{dom} F)$
- $(F \circ R)$: $D_1 \to \mathbb{Q}$ of same type as F
- $F(S)$: $\mathbb{Q}$ of same type as F

if R is single-valued, then sum/bound is computed over a single point

Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

$$F = \{ [i,j] \to ij + i - N + 1 \}$$

$$I = \{ [i,j] : 0 \leq i < N \wedge i \leq j < N \}$$

$$F(I) = \left\{ \frac{5}{12}N - \frac{1}{8}N^2 - \frac{5}{12}N^3 + \frac{1}{8}N^4 : N \geq 1 \right\}$$

Pointer Conversion

```
p = a;
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p += 1 + j * ((j-i)/4);
        *p = hard_work(i,j);
    }
```

Can we parallelize this code?

Pointer Conversion

```
p = a;
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p += 1 + j * ((j-i)/4);
        *p = hard_work(i,j);
    }
```

Can we parallelize this code?

$\Rightarrow$ No, (false) dependency through p

$\Rightarrow$ Compute closed formula for p $\quad p = a + \sum_{\substack{(i',j') \in I \\ (i',j') \preccurlyeq (i,j)}} 1 + j' \left\lfloor \frac{j' - i'}{4} \right\rfloor$

$$I = \{ [i,j] : 0 \leq i < N \wedge i \leq j < N \}$$

$$F = \left\{ [i,j] \to 1 + j \left\lfloor \frac{j - i}{4} \right\rfloor \right\}$$

$F \circ (I \succcurlyeq I) = \ldots$

Nested Relations

Assume that we want to count the number of statement instances between any given pair of statement instances

- $\Rightarrow$ we need to create a relation between a pair of statement instances and a third statement instance
- $\Rightarrow$ nested relations

Nested Relations

Assume that we want to count the number of statement instances between any given pair of statement instances

$\Rightarrow$ we need to create a relation between a pair of statement instances and a third statement instance

$\Rightarrow$ nested relations

Example:

$$\{ (S[\mathbf{i}] \to S[\mathbf{j}]) \to S[\mathbf{k}] : \mathbf{i} \prec \mathbf{k} \prec \mathbf{j} \}$$

1D:

$$\begin{aligned} &\mathrm{card}\{ [[i] \to [j]] \to [k] : 0 \leq i < k < j \leq n \} \\ = &\{ [[i] \to [j]] \to j - i - 1 : 0 \leq i \wedge i + 2 \leq j \leq n \} \end{aligned}$$

[10]

Wrapping, Unwrapping, Domain Map and Range Map

Given a set S and a binary relation R

- Wrap (`iscc: wrap`)

  $\mathcal{W}R = \{ [\mathbf{i} \to \mathbf{j}] : \mathbf{i} \to \mathbf{j} \in R \}$

[10]

Wrapping, Unwrapping, Domain Map and Range Map

Given a set S and a binary relation R

- Wrap (`iscc: wrap`)

  $\mathcal{W}R = \{ [\mathbf{i} \to \mathbf{j}] : \mathbf{i} \to \mathbf{j} \in R \}$

- Unwrap (`iscc: unwrap`)

  $\mathcal{W}^{-1}S = \{ \mathbf{i} \to \mathbf{j} : \exists n : n[\mathbf{i} \to \mathbf{j}] \in S \}$

[10]

Wrapping, Unwrapping, Domain Map and Range Map

Given a set S and a binary relation R

- Wrap (`iscc: wrap`)

  $\mathcal{W}R = \{ [\mathbf{i} \to \mathbf{j}] : \mathbf{i} \to \mathbf{j} \in R \}$

- Unwrap (`iscc: unwrap`)

  $\mathcal{W}^{-1}S = \{ \mathbf{i} \to \mathbf{j} : \exists n : n[\mathbf{i} \to \mathbf{j}] \in S \}$

- Domain projection (`iscc: domain_map`)

  $\underrightarrow{\mathrm{dom}}\, R = \{ [\mathbf{i} \to \mathbf{j}] \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R \}$

[10]

Wrapping, Unwrapping, Domain Map and Range Map

Given a set S and a binary relation R

- Wrap (`iscc: wrap`)

  $\mathcal{W}R = \{ [\mathbf{i} \to \mathbf{j}] : \mathbf{i} \to \mathbf{j} \in R \}$

- Unwrap (`iscc: unwrap`)

  $\mathcal{W}^{-1}S = \{ \mathbf{i} \to \mathbf{j} : \exists n : n[\mathbf{i} \to \mathbf{j}] \in S \}$

- Domain projection (`iscc: domain_map`)

  $\underrightarrow{\mathrm{dom}}\, R = \{ [\mathbf{i} \to \mathbf{j}] \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R \}$

- Range projection (`iscc: range_map`)

  $\underrightarrow{\mathrm{ran}}\, R = \{ [\mathbf{i} \to \mathbf{j}] \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in R \}$

[10]

Spaces

The elements of a set are of the form

- $n[i_0, i_1, \ldots, i_{d-1}]$, with S1[0, 1]
  - n an identifier
  - $d \geq 0$
  - i_j integers

Define

Space The space $\mathcal{S}\mathbf{i}$ of an element $\mathbf{i}$ is

- n/d S1/2

Values The value vector $\mathcal{V}\mathbf{i}$ of an element $\mathbf{i}$ is

- $(i_0, i_1, \ldots, i_{d-1})$ (0, 1)

[10]

Spaces

The elements of a set are of the form

- $n[i_0, i_1, \ldots, i_{d-1}]$, with — $\mathtt{S1}[0, 1]$
  - n an identifier
  - $d \geq 0$
  - i_j integers
- $n[\mathbf{j} \to \mathbf{k}]$, with — $[\mathtt{S1}[0, 1] \to \mathtt{S2}[3]]$
  - n an identifier
  - $\mathbf{j}$, $\mathbf{k}$ elements

Define

Space The space $\mathcal{S}\mathbf{i}$ of an element $\mathbf{i}$ is

- n/d — $\mathtt{S1}/2$
- $(n, \mathcal{S}(\mathbf{j}), \mathcal{S}(\mathbf{k}))$ — $(\cdot, \mathtt{S1}/2, \mathtt{S2}/1)$

Values The value vector $\mathcal{V}\mathbf{i}$ of an element $\mathbf{i}$ is

- $(i_0, i_1, \ldots, i_{d-1})$ — $(0, 1)$
- $\mathcal{V}(\mathbf{j}) \| \mathcal{V}(\mathbf{k})$ — $(0, 1, 3)$

[2]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                    /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();            /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];      /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();            /*S6*/
  return arrB;
}
```

[2]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                          /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();                  /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];      /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();                  /*S6*/
  return arrB;
}
```

$$I = \{ \mathrm{m0}[m] \rightarrow \mathrm{S1}[c] : 0 \leq c < m;$$
$$\mathrm{m0}[m] \rightarrow \mathrm{S2}[c] : 0 \leq c < m;$$
$$\mathrm{m1}[k] \rightarrow \mathrm{S3}[i] : 1 \leq i \leq k;$$
$$\mathrm{m1}[k] \rightarrow \mathrm{S4}[i] : 1 \leq i \leq k;$$
$$\mathrm{m2}[n] \rightarrow \mathrm{S5}[];$$
$$\mathrm{m2}[n] \rightarrow \mathrm{S6}[j] : 1 \leq j \leq n \}$$

[2]

Dynamic Memory Requirement Estimation

How much (scoped) memory is needed?

$\Rightarrow$ compute for each method

$\mathrm{ret}_{\mathtt{m}}$ size of memory returned by `m`

$\mathrm{cap}_{\mathtt{m}}$ size of memory "captured" (not returned) by `m`

$\mathrm{memRq}_{\mathtt{m}}$ total memory requirements of `m`

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

[2]

Dynamic Memory Requirement Estimation

How much (scoped) memory is needed?

$\Rightarrow$ compute for each method

$\mathrm{ret}_{\mathtt{m}}$ size of memory returned by $\mathtt{m}$

$\mathrm{cap}_{\mathtt{m}}$ size of memory "captured" (not returned) by $\mathtt{m}$

$\mathrm{memRq}_{\mathtt{m}}$ total memory requirements of $\mathtt{m}$

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

$\Rightarrow$ summarize over iteration domain, i.e., compose with $M = (\underrightarrow{\mathrm{dom}}\, I)^{-1}$

$$\begin{aligned} M = \{ & \mathtt{m0}[m] \to [\mathtt{m0}[m] \to \mathtt{S1}[c]] : 0 \le c < m; \\ & \mathtt{m0}[m] \to [\mathtt{m0}[m] \to \mathtt{S2}[c]] : 0 \le c < m; \\ & \mathtt{m1}[k] \to [\mathtt{m1}[k] \to \mathtt{S3}[i]] : 1 \le i \le k; \\ & \mathtt{m1}[k] \to [\mathtt{m1}[k] \to \mathtt{S4}[i]] : 1 \le i \le k; \\ & \mathtt{m2}[n] \to [\mathtt{m2}[n] \to \mathtt{S5}[]]; \mathtt{m2}[n] \to [\mathtt{m2}[n] \to \mathtt{S6}[j]] : 1 \le j \le n \} \end{aligned}$$

[2]

Dynamic Memory Requirement Estimation

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

[2]

Dynamic Memory Requirement Estimation

$$\mathrm{ret_m} + \mathrm{cap_m} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret_p}$$

$$\mathrm{memRq_m} = \mathrm{cap_m} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq_p}$$

```
B[] m2(int n) {
 B[] arrB = new B[n];        /*S5*/
 for (j = 1; j <= n; j++)
  B b = new B();             /*S6*/
 return arrB;
}
```

[2]

Dynamic Memory Requirement Estimation

$$\mathrm{ret_m} + \mathrm{cap_m} = \sum_{\mathtt{p}\ \text{called by}\ \mathtt{m}} \mathrm{ret_p}$$

$$\mathrm{memRq_m} = \mathrm{cap_m} + \max_{\mathtt{p}\ \text{called by}\ \mathtt{m}} \mathrm{memRq_p}$$

```
B[] m2(int n) {
 B[] arrB = new B[n];          /*S5*/
 for (j = 1; j <= n; j++)
  B b = new B();               /*S6*/
 return arrB;
}
```

$$\mathrm{ret_{m2}} = \{\, [\mathtt{m2}[n] \to \mathtt{S5}[]] \to n : n \geq 0 \,\} \circ M$$

$$\mathrm{cap_{m2}} = \{\, [\mathtt{m2}[n] \to \mathtt{S6}[j]] \to 1 \,\} \circ M$$

$$\mathrm{memRq_{m2}} = \mathrm{cap_{m2}} + \{\, \mathtt{m2}[n] \to \max(0) \,\}$$

[2]

Dynamic Memory Requirement Estimation

$$\mathrm{ret_m} + \mathrm{cap_m} = \sum_{\text{p called by m}} \mathrm{ret_p}$$

$$\mathrm{memRq_m} = \mathrm{cap_m} + \max_{\text{p called by m}} \mathrm{memRq_p}$$

```
B[] m2(int n) {
 B[] arrB = new B[n];        /*S5*/
 for (j = 1; j <= n; j++)
  B b = new B();             /*S6*/
 return arrB;
}
```

$$\mathrm{ret_{m2}} = \{\, [\mathtt{m2}[n] \to \mathtt{S5}[]] \to n : n \geq 0 \,\} \circ M = \{\, \mathtt{m2}[n] \to n : n \geq 0 \,\}$$

$$\mathrm{cap_{m2}} = \{\, [\mathtt{m2}[n] \to \mathtt{S6}[j]] \to 1 \,\} \circ M = \{\, \mathtt{m2}[n] \to n : n \geq 1 \,\}$$

$$\mathrm{memRq_{m2}} = \mathrm{cap_{m2}} + \{\, \mathtt{m2}[n] \to \max(0) \,\} = \{\, \mathtt{m2}[n] \to \max(n) : n \geq 1 \,\}$$

[2]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                  /* S3 */
        B[] dummyArr = m2(i);           /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$\mathrm{ret}_{\mathtt{m2}}$ is a function of the arguments of `m2`

We want to use it as a function of the arguments and local variables of `m1`

[2]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                  /* S3 */
        B[] dummyArr = m2(i);           /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$\mathrm{ret}_{\mathtt{m2}}$ is a function of the arguments of `m2`

We want to use it as a function of the arguments and local variables of `m1`

$\Rightarrow$ define parameter binding

[2]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                          /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();                  /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];              /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();                  /*S6*/
  return arrB;
}
```

$$I = \{ \mathtt{m0}[m] \rightarrow \mathtt{S1}[c] : 0 \leq c < m;$$
$$\mathtt{m0}[m] \rightarrow \mathtt{S2}[c] : 0 \leq c < m;$$
$$\mathtt{m1}[k] \rightarrow \mathtt{S3}[i] : 1 \leq i \leq k;$$
$$\mathtt{m1}[k] \rightarrow \mathtt{S4}[i] : 1 \leq i \leq k;$$
$$\mathtt{m2}[n] \rightarrow \mathtt{S5}[];$$
$$\mathtt{m2}[n] \rightarrow \mathtt{S6}[j] : 1 \leq j \leq n \}$$

[2]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                          /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();                  /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];              /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();                  /*S6*/
  return arrB;
}
```

$$
\begin{aligned}
I = \{ & \mathtt{m0}[m] \rightarrow \mathtt{S1}[c] : 0 \leq c < m; \\
& \mathtt{m0}[m] \rightarrow \mathtt{S2}[c] : 0 \leq c < m; \\
& \mathtt{m1}[k] \rightarrow \mathtt{S3}[i] : 1 \leq i \leq k; \\
& \mathtt{m1}[k] \rightarrow \mathtt{S4}[i] : 1 \leq i \leq k; \\
& \mathtt{m2}[n] \rightarrow \mathtt{S5}[]; \\
& \mathtt{m2}[n] \rightarrow \mathtt{S6}[j] : 1 \leq j \leq n \}
\end{aligned}
$$

$$
\begin{aligned}
B^{\mathtt{m0}} = \{ & [\mathtt{m0}[m] \rightarrow \mathtt{S1}[c]] \rightarrow \mathtt{m1}[c]; \\
& [\mathtt{m0}[m] \rightarrow \mathtt{S2}[c]] \rightarrow \mathtt{m2}[2m - c] \} \\
B^{\mathtt{m1}} = \{ & [\mathtt{m1}[k] \rightarrow \mathtt{S4}[i]] \rightarrow \mathtt{m2}[i] \}
\end{aligned}
$$

[2]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                  /* S3 */
        B[] dummyArr = m2(i);           /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$\mathrm{ret}_{\mathtt{m2}}$ is a function of the arguments of `m2`

We want to use it as a function of the arguments and local variables of `m1`

$\Rightarrow$ define parameter binding

[2]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                  /* S3 */
        B[] dummyArr = m2(i);           /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$$\mathrm{ret}_{\mathtt{m1}} = \{\, \mathtt{m1}[k] \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m1}} = \left(\{\, [\mathtt{m1}[k] \to \mathtt{S3}[i]] \to 1 \,\} + \mathrm{ret}_{\mathtt{m2}} \circ B^{\mathtt{m1}}\right) \circ M$$

$$\mathrm{memRq}_{\mathtt{m1}} = \mathrm{cap}_{\mathtt{m1}} + \left(\mathrm{memRq}_{\mathtt{m2}} \circ B^{\mathtt{m1}} \circ M\right)$$

[2]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                    /* S3 */
        B[] dummyArr = m2(i);             /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$$\mathrm{ret}_{\mathtt{m1}} = \{\, \mathtt{m1}[k] \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m1}} = \left(\{\, [\mathtt{m1}[k] \to \mathtt{S3}[i]] \to 1 \,\} + \mathrm{ret}_{\mathtt{m2}} \circ B^{\mathtt{m1}}\right) \circ M$$

$$= \left\{ \mathtt{m1}(k) \to \frac{3}{2}k + \frac{1}{2}k^2 : k \ge 1 \right\}$$

$$\mathrm{memRq}_{\mathtt{m1}} = \mathrm{cap}_{\mathtt{m1}} + \left(\mathrm{memRq}_{\mathtt{m2}} \circ B^{\mathtt{m1}} \circ M\right)$$

$$= \left\{ \mathtt{m1}(k) \to \max\left(\frac{5}{2}k + \frac{1}{2}k^2\right) : k \ge 1 \right\}$$

[2]

Dynamic Memory Requirement Estimation

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                              /* S1 */
        B[] m2Arr = m2(2 * m - c);          /* S2 */
    }
}
```

$$B^{\mathtt{m0}} = \{\, [\mathtt{m0}[k] \to \mathtt{S1}[c]] \to \mathtt{m1}[c]; [\mathtt{m0}[k] \to \mathtt{S2}[c]] \to \mathtt{m2}[2m - c] \,\}$$

$$\mathrm{ret}_{\mathtt{m0}} = \{\, \mathtt{m0}[m] \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m0}} = (\mathrm{ret}_{\mathtt{m1}} + \mathrm{ret}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M$$

$$\mathrm{memRq}_{\mathtt{m0}} = \mathrm{cap}_{\mathtt{m0}} + ((\mathrm{memRq}_{\mathtt{m1}} + \mathrm{memRq}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M)$$

[2]

Dynamic Memory Requirement Estimation

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                                  /* S1 */
        B[] m2Arr = m2(2 * m - c);              /* S2 */
    }
}
```

$$B^{\mathtt{m0}} = \{\, [\mathtt{m0}[k] \to \mathtt{S1}[c]] \to \mathtt{m1}[c]; [\mathtt{m0}[k] \to \mathtt{S2}[c]] \to \mathtt{m2}[2m - c] \,\}$$

$$\mathrm{ret}_{\mathtt{m0}} = \{\, \mathtt{m0}[m] \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m0}} = (\mathrm{ret}_{\mathtt{m1}} + \mathrm{ret}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M$$

$$= \left\{ \mathtt{m0}[m] \to \frac{1}{2}m + \frac{3}{2}m^2 : m \geq 1 \right\}$$

$$\mathrm{memRq}_{\mathtt{m0}} = \mathrm{cap}_{\mathtt{m0}} + ((\mathrm{memRq}_{\mathtt{m1}} + \mathrm{memRq}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M)$$

$$= \left\{ \mathtt{m0}[m] \to \max\left(-2 + 2m + 2m^2, \frac{5}{2}m + \frac{3}{2}m^2\right) : m \geq 1 \right\}$$

References I

[1] Kristof Beyls and Erik D'Hollander. "Generating Cache Hints for Improved Program Efficiency". In: *Journal of Systems Architecture* 51.4 (2005), pp. 223–250. DOI: 10.1016/j.sysarc.2004.09.004.

[2] Philippe Clauss, Federico Javier Fernandez, Diego Garbervetsky, and Sven Verdoolaege. "Symbolic polynomial maximization over convex sets and its application to memory requirement estimation". In: *IEEE Transactions on VLSI Systems* 17.8 (Aug. 2009), pp. 983–996. DOI: 10.1109/TVLSI.2008.2002049.

[3] Paul Feautrier and Christian Lengauer. "The Polyhedron Model". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Springer, 2011, pp. 1581–1592.

[4] Tobias Grosser, Armin Größlinger, and Christian Lengauer. "Polly - Performing polyhedral optimizations on a low-level intermediate representation". In: *Parallel Processing Letters* 22.04 (2012). DOI: 10.1142/S0129626412500107.

References II

[5] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and David Wonnacott. *The Omega Library*. Tech. rep. University of Maryland, Nov. 1996.

[6] Vincent Loechner and Doran K. Wilde. "Parameterized Polyhedra and Their Vertices". In: *International Journal of Parallel Programming* 25.6 (Dec. 1997), pp. 525–549. DOI: 10.1023/A:1025117523902.

[7] Sven Verdoolaege. "isl: An Integer Set Library for the Polyhedral Model". In: *Mathematical Software - ICMS 2010*. Ed. by Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama. Vol. 6327. Lecture Notes in Computer Science. Springer, 2010, pp. 299–302. DOI: 10.1007/978-3-642-15582-6_49.

References III

[8] Sven Verdoolaege. "Counting Affine Calculator and Applications". In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*. Chamonix, France, Apr. 2011. DOI: 10.13140/RG.2.1.2959.5601.

[9] Sven Verdoolaege. "Polyhedral process networks". In: *Handbook of Signal Processing Systems*. Ed. by Shuvra Bhattacharrya, Ed Deprettere, Rainer Leupers, and Jarmo Takala. 2nd ed. Springer, 2013, pp. 1335–1375. DOI: 10.1007/978-1-4614-6859-2_41.

[10] Sven Verdoolaege. *Presburger Formulas and Polyhedral Compilation*. 2016.

[11] Sven Verdoolaege and Tobias Grosser. "Polyhedral Extraction Tool". In: *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*. Paris, France, Jan. 2012. DOI: 10.13140/RG.2.1.4213.4562.

References IV

[12] Sven Verdoolaege, Gerda Janssens, and Maurice Bruynooghe. "Equivalence Checking of Static Affine Programs Using Widening to Handle Recurrences". In: *ACM Trans. Program. Lang. Syst.* 34.3 (Nov. 2012), 11:1–11:35. DOI: 10.1145/2362389.2362390.

[13] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. "Polyhedral parallel code generation for CUDA". In: *ACM Trans. Archit. Code Optim.* 9.4 (2013), p. 54. DOI: 10.1145/2400682.2400713.

[14] Sven Verdoolaege, R. Seghir, K. Beyls, Vincent Loechner, and Maurice Bruynooghe. "Counting integer points in parametric polytopes using Barvinok's rational functions". In: *Algorithmica* 48.1 (June 2007), pp. 37–66. DOI: 10.1007/s00453-006-1231-0.

[15] Doran K. Wilde. *A Library for doing polyhedral operations*. Tech. rep. 785. IRISA, Rennes, France, 1993, 45 p.